pl/2d: A Programming Language for Generating 2-Dimensional Patterns

by

Karen Sebela Lantz

May 1974

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-74-647

pl/2d: A Programming Language for Generating 2-Dimensional Patterns

by

Karen Sebela Lantz

May 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported in part by the Alfred P. Sloan Foundation.

pl/2d: A PROGRAMMING LANGUAGE FOR GENERATING 2-DIMENSIONAL PATTERNS

BY

KAREN SEBELA LANTZ

B.S., University of Illinois, 1971

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1974

Urbana, Illinois

TABLE OF CONTENTS

I. INTRODUCTION

There probably is no better way to teach introductory computer science concepts than to use a computer system with a flexible graphics terminal to present educational material. Both textual instruction and interactive simulations can demonstrate a particular concept more effectively than a lecturer before a huge classroom audience. Pl/2d has been designed to demonstrate to students one particular topic, recursion, in a simple and understandable way.

Pl/2d is a miniprogramming language for generating 2-dimensional patterns that graphically demonstrates the process of recursion to a beginning programmer. Hopefully, any person with an elementary background in programming and high school algebra can be successfully programming in this mini-language within one hour; and in two or three hours time s(he) should be adept enough with recursive programming to be able to use recursion in any other language for which the student has a programming manual. The aim of designing pl/2d is to have a tool that is not complicated to learn or use, and that easily generates examples at the student's direction. Pl/2d might be used in a computer science course just as a ripple tank experiment is used to demonstrate wave theory in a physics course. The theory of recursion could be by-passed at first by introducing the concept informally and by trying to establish a firm basis for rigorous proof later.

This thesis has been written to explain the reasons why the language was designed and to more fully document the language itself. The program

has been designed as part of a series of lessons for a computer appreciation course written under the direction of Professor Jurg Nievergelt and funded by the Alfred P. Sloan Foundation. The course is being implemented on PLATO IV, a computer-assisted instructional system developed by the Computer-Based Education Research Laboratory at the University of Illinois [1].

II. MOTIVATION

The concept of recursion in programming is difficult to explain to an unsophisticated audience. The natural way to expose students to the concept is through various examples. A favorite non-mathematical one is the painter painting a picture of himself painting a picture, ..., etc. The telescoping effectively gives the student an easy way to "see" recursion work. An explanation of an ending condition might be the painter repeating this process until the picture is too small to paint.

This example is useful because the output exhibits the process of recursion. The output of most recursive programming examples, such as a program that calculates a factorial, does not supply any information to prove whether the program was written iteratively or recursively. However, writing the non-mathematical example algorithmically in mathematical notation is impossible; and eventually the instructor wants to eliminate non-mathematical examples and expand the programming possibilities.

Currently, one language that is often used to demonstrate recursion is PL/1. However, there are drawbacks to using this language. The language is sufficiently complex that many features must be taught before recursive examples can be introduced. Learning the language overshadows learning the programming techniques. By the time recursion is presented, it is at the end of the course of instruction, and the instructor's presentations are forced to be cursory. Any lasting impact that recursion might have made will be diminished merely by lack of both the instructor's

and student's time. Even so, the student might realize the importance of recursion, but still not understand why problems could not be solved in other ways. Another drawback is that the output of PL/1 does not exhibit any reason why recursion was used to solve the problem.

A smaller language would eliminate some of the problem. A language with graphical output of a recursive program could facilitate an easier grasp of the function of such programming; and finally, implementing the language on an interactive system gives the student an opportunity to immediately edit her/his program.

Smaller packages of programming languages are a usable way of teaching complex concepts and processes. These mini-languages can eliminate the double risk of having to learn two complex things at once, a powerful language plus a difficult concept. A mini-language would have its usefulness only as a learning tool and not as a production vehicle. After a student understood the concept singularly demonstrated, then the language could be discarded. Recursive programming techniques learned by using pl/2d are easily transferable to other languages because a student does not have to contend with irrelevant techniques in pl/2d. Learning should also occur faster because there is the ability to practice a technique, plus the reward of seeing the effects of the program immediately. There is no frustration because the results are visible to the student.

III. DESIGN

The design of the entire lesson is divided into three main sections. First, the language is described. Pl/2d allows the students to draw any 2-dimensional pattern with dots, line segments, circles, and arcs. A student may draw a simple pattern with no recursion, but, of course, the more interesting patterns require that the student program recursively.

The definition of the language includes three types of commands: 1) graphing instructions to produce dots, line segments, connected line segments, circles, and arcs; 2) instructions to define the beginning and end of the program, as well as a statement that initiates the repetition of a program; and, 3) a decision instruction that allows different paths to be taken within the program during execution.

Secondly, there is the compilation-editor and interpreter for pl/2d. Editing functions such as inserting, deleting, and replacing are included with operation functions such as destroying a program or initiating execution. These allow the student to write, debug, and execute her/his program.

The third part of the project is a lesson utilizing the interpreter to demonstrate all the functions of pl/2d. The lesson not only teaches the syntax of the language but, through examples, presents ideas for types of programs that can be written in pl/2d.

A. Syntax

Two goals were chosen in defining pl/2d: 1) that the language would be a viable tool for demonstrating recursion without offering alternative

techniques for programming; and 2) that the syntax should not be difficult to learn. The statement types of pl/2d are similar to statements in other languages so that there is no confusion when transferring to another language. The language consists of seven statements. Thus the student should be able to learn the language in about one-half hour and be able to competently program that same hour.

The graphical instructions are defined by the environment of the terminal. The PIATO terminal is a 512 x 512 matrix of independently addressable points. The graphing commands draw lines and arcs by activating points. The matrix is treated as a cartesian graph with an x and y-axis and an origin of (0, 0) in the left-bottom corner of the terminal screen.

1. Draw Command

The draw command specifies a point to light by naming the x and y-coorindate of the point

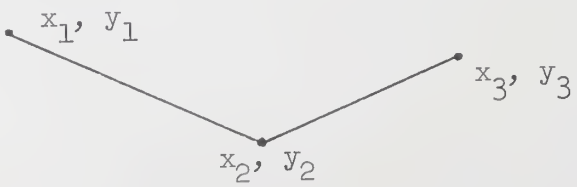
draw x, y.

A line segment is just a connection between 2 specified points,

draw x_1, y_1, x_2, y_2

and a series of line segments are connected if a series of points are specified by the draw command.

draw x_1, y_1, x_2, x_3, y_3



The general case of the draw commands allows 8 arguments, which will draw 3 connected line segments.

draw x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4

2. Circle Command

Circles and arcs are drawn with the circle command. A radius length and center-point specify the dimensions for the circle.

circle radius, x-center, y-center

To draw an arc one specifies two more arguments in the circle command, the beginning and ending angle of the arc measured in degrees.

circle radius, x-center, y-center, begin-angle, end-angle

3. Program Definition Statement

The program definition names the program and declares any parameters to be initialized immediately before execution.

program name-of-program $param_1$, $param_2$, ..., $param_8$

The name of the program may be any combination of small-case letters and digits from length 2 to 10 starting with a letter. There may be up to 8 parameters specified to be used in the program. The names of the parameters must be a single letter of the alphabet from s thru z. The parameters defined in the definition are those expected to be used in the body of the program.

4. End Statement

The end of the program is specified by the end statement. This statement causes a return to the calling program during execution, or ends execution. The syntax of the statement is merely the keyword 'end'.

5. Program Use Statement

A program can be reiterated by using the program use statement. This command saves a pointer to the next statement to be executed, saves all the values of the parameters defined in the program definition, and starts executing the program at the beginning. The syntax of the program use statement is

name-of-program arg 1, arg 2, ..., arg 8

The number of arguments must be the same as the number in the program definition. When this statement is executed the value of each argument is assigned to the corresponding parameter in the definition. An argument is defined by any arithmetic or logical expression as defined in Table 1. The expressions used as arguments in the program use statement can also be used in the parameters of any of the graphical commands.

6. Case Statement

The case statement is the conditional statement in pl/2d. Different paths may be taken during execution time. A list of conditions is listed serially. During execution the first true condition that is found determines the path to be taken. After each condition any number of pl/2d statements may follow. Those statements after the <u>first</u> true condition are executed. All other conditions as well as the statements following those conditions are ignored. The syntax of the case statement is

```
case
condition_1
   <pl/2d statements>
condition_2
    .
    .
    .
condition_i
    .
    .
    .
esac
```

Table 1

Bachnaus-naur Description of Expressions in pl/2d

<exp>	::=	<arith> | <logical>
<logical>	::=	<logical> <operator1> <logical> |
		<arith> <operator2> <arith>
<arith>	::=	<mult> | <mult>
<mult>	::=	<add> [(x 1 ÷) <arith>]
<add>	::=	<simple> [(+ 1 -) <simple>]
<simple>	::=	<func> | $(_r$ <arith> $)_r$ | $(_r$ <logical> $)_r$ |
		[-] <number> | [-] <var>
<func>	::=	abs $(_r$ <exp> $)_r$ |
		sin $(_r$ <exp> $)_r$ |
		cos $(_r$ <exp> $)_r$ |
		sqrt $(_r$ <exp> $)_r$
<number>	::=	{<digit>} [. {<digit>}]
<digit>	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<operator1>	::=	$ and $ | $ or $
<operator2>	::=	< | > | ≤ | ≥ | = | ≠
<var>	::=	s | t | u | v | w | x | y | z

note: $(_r$ and $)_r$ are real parentheses

(and) are meta-parentheses

The case statement is delimited by the keyword 'case' at the beginning and by the keyword 'esac' at the end. The syntax is similar to the LISP cond statement on which pl/2d is based [2]. A condition is defined by a logical expression. In the implementation of pl/2d, a true value is associated with -1 and a false value with 0. The arithmetic value of -1 is also accepted as an unconditional choice that will always be chosen if reached.

B. The Editor/Compiler and Interpreter

The editor and compiler of pl/2d are really one program. As soon as the student inputs a command it is compiled and stored if correct. At any step of writing the program, the student is assured that all previous lines are meaningful. Since pl/2d was implemented on an interactive system it was feasible to implement the compilation this way. Immediate feedback for the student reinforces learning the correct syntax of the language. Pl/2d recognizes every syntax error and responds with an appropriate message. The student must fix the error or delete the statement to precede. Besides any editing features programmed, the PLATO terminal has many built-in functions that help the student correct a line quickly and easily. Also, at any time, a small review of the editing commands and the pl/2d language is available to the student. Hopefully, the error message and this reference materials will be enough information for the student to correct her/his errors.

The editor allows a student to write and edit a program. The editing commands are recognized as keywords like pl/2d. Then, if necessary, a statement number and number of statements to be changed follow the keyword. The commands are described in Table 2. There is a maximum of 25 lines accepted as the total length of the program.

T le 2

insert n	start inserting after line n
replace n	start replacing statements with line n
delete n, m	delete m statements, starting at line n
destory	destroy the entire program
run	execute the program

The functions of inserting and replacing will be continued until the length of the program exceeds this limit or a blank line is accepted by the editor. A blank line shifts the editor back into normal mode, which allows adding statements to the end of the program.

The program is only compiled up to the point where the student is editing. Thus if a student decides to delete line 3 of a 12-line program, the program will only be compiled up to line 3 before the deletion and the rest of the program must be recompiled after the deletion. The recompilation of the program constantly makes sure that the student understands where her/his errors are in the program.

When the student elects to execute his program, control is passed to the interpreter. If the student has defined any parameters in the program definition, the interpreter now asks for initial values. The student is also queried if (s)he would like a trace of the program. The trace is an arrow that points to the statement executing at the moment.

Recursion is implemented using a stack. Any call to the program results in all the values being saved as well as the return address. The only execution error that may occur is if the stack overflows. At that

time execution is halted, and the student is asked to re-edit her/his program. If asked for, a small explanation of what happened is given.

One point to consider is that often a graphical command will reference some place off the screen. This is reasonable sometimes for drawing arcs only partially on the screen since it is impossible to guess exactly which degree will take the arc off the screen. But if the program completely misses the screen, it is obviously not correct. These two situations are both transparent to the simulator. Pl/2d is simulated by TUTOR commands; and these commands will generate points off the screen and light them, even if only on an imaginary extension of the defined matrix. To partially remedy this, a student can always prematurely stop the execution of her/his program. No execution time diagnostics are given for this condition, however. It is assumed that the student is intelligent enough to stop execution.

1. Data Structure

A conventional approach has been taken in compiling the language. Most of the time a student spends programming in pl/2d will be spent editing a program. Thus the intermediate code is kept as close to the original as possible. A fixed length of memory is defined for each command. The command is translated into a token number and the length of each parameter is stored. The parameters themselves are stored in strings and not as compiled code. Each time a pl/2d command is executed, the parameters of that command are compiled. This approach was chosen because only a small fraction of the time spent on one program will be during execution. However, this might not have been wise because of the number of recursions and the large number of parameters used in some patterns. Execution might be slower

even though the parameters are compiled relatively quick by a specail TUTOR command.

The PLATO IV system has 60-bit words and 6-bit characters. Each statement in pl/2d was stored in a 6-variable data structure. A fixed length was chosen because it is easily accessed and saves time instead of searching for a variable length command. It was necessary to use a fixed length to accommodate a maximum length program.†

Figure 1 is an example of the data structure used to compile a statement in pl/2d.

2. Problems

Two problems occurred in implementing pl/2d. First, the compiler could not recognize the difference between the name of a program and either the variables or keywords used in that program. The solution chosen partially solves the problem. All program names must be greater than 1 character in length. However, the compiler does not recognize when a program name is identical to a keyword. The choice of keyword is always matched first. Thus, the student is warned that program names should not be identical to keywords.

Another problem of a pl/2d program is slow execution; more complex arguments and larger amounts of recursion visibly slows execution.‡

---

†The amount of memory allocated to a PLATO program is a non-executable function.

‡Although the trend of PLATO IV programming leans toward lessons that more CPU utilization, the time allotted each terminal per time slice by the system is decreasing. As the lesson is visibly slower, the student might lose interest. The needs of the programs being written demand that the design of computer-assisted instructional system, and PLATO IV in particular, must change. Educational uses of computers do not necessarily imply minimal demands being placed on the system.

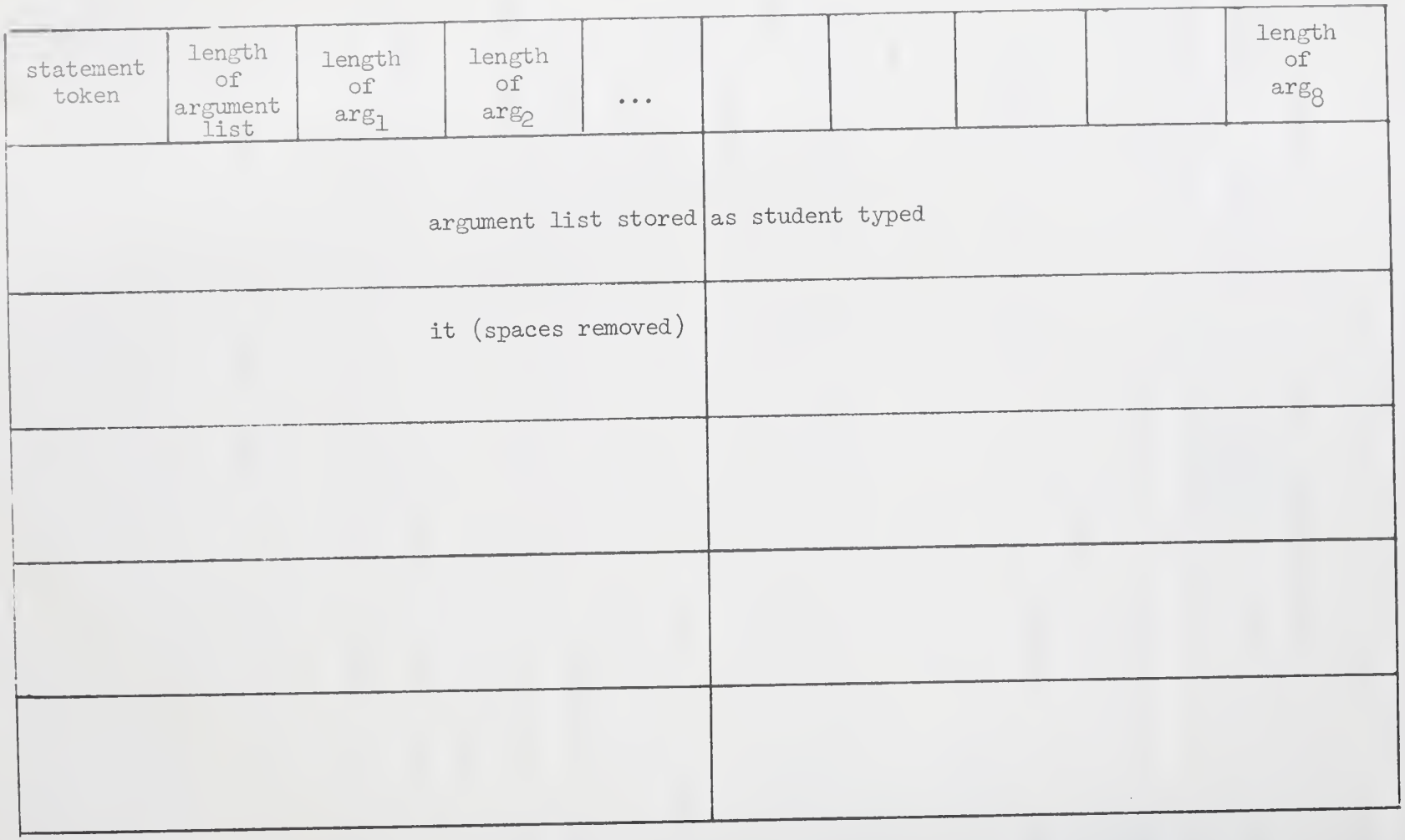


Figure 1. Data Structure for a Statement in pl/2d

C. The Lesson

Besides presenting the syntax of the language, the lesson demonstrates some simple examples for the student. Both code and the execution of the programs are presented for each new statement that is presented to the student.

As an example, after the graphing and program definition statements are introduced, the student is presented with a program that draws the pattern shown in Figure 2.

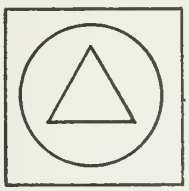


Figure 2. Pattern Drawn by Program Emblem

From this example the student is introduced to the use of parameters in the program definition and how, by changing the values of these parameters, the design can be drawn in different positions on the screen or even be programmed to change size.

The student is first shown a program with fixed arguments (Example 1). Gradually the student is guided through intermediate steps until the final version of the program that changes position and size is presented (Example 2).

A second example introduces the case statement. The student is asked to choose a starting value between 1 and 3 for a program. The outcome from her/his choice is a face with either a smile, frown, or surprised expression.

```
program   emblem 1
draw      400, 350, 450, 350, 450, 400
draw      450, 400, 400, 400, 400, 350
circle    25, 425, 375
draw      410, 360, 440, 360, 425, 390, 410, 360
end
```

Example 1. Program Emblem 1

```
program   emblem 3   s, t, u
draw      s, t, s+u, t, s+u, t+u
draw      s+u, t+u, s, t+u, s, t
circle    u÷2, s+u÷2, t+u÷2
draw      s+u÷5, t+u÷5, s-u÷5+u, t+u÷5, s+u÷2, t-u÷5+u
draw      s+u÷2, t-u÷5+u, s+u÷5, t+u÷5
end
```

Example 2. Final program of emblem which changes position and size. (Line 5 of Example 1 has been expanded to 2 statements because the editor only allows 50 characters, including spaces, per line.)

The student can execute this program any number of times before s(he) sees the actual code.

The program use statement is demonstrated at first with an iterative program. A sunburst is drawn for the student while the program's execution is traced on the screen. Iterative programming is probably the first serious attempt the student will try on her/his own. The lesson stresses the point that the case statement must be used. With the program statement the program will never stop until the stack overflows. A correct (Example 3) as well as an incorrect (Example 4) version of the program is presented.

After a gradual introduction to programs in pl/2d, a recursive example is given. The program called 'tri' draws three triangles in every other triangle until the inside triangles have a base less than 10 dots wide. The final result of the program is pictured in Figure 3.

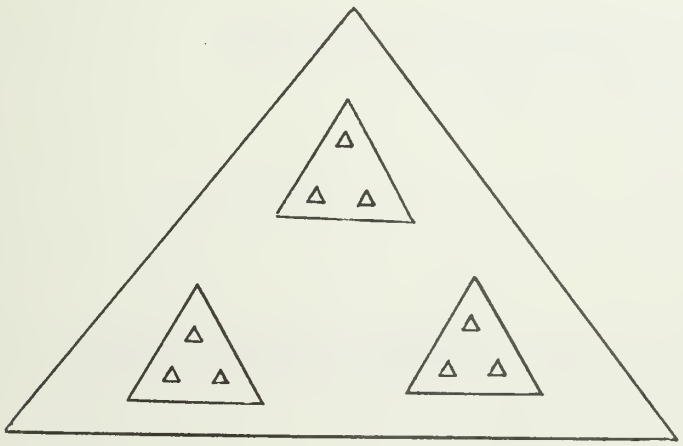


Figure 3. Final Output of Recursive Program Tri

```
program    sunburst s
draw   256, 256, 50 x sin(s) + 256, 50 x cos(s) + 256
sunburst   s+10
end
```

Example 3. Incorrect example of iterative program

```
program    sunburst s
draw   256, 256, 50 x sin(s) + 256, 50 x cos(s) + 256
case
     s < 360
sunburst   s+10
esac
end
```

Example 4. Correct example of iterative program case statement provides for end of execution

The student sees the order the program is executed by watching a trace; (Example 5) and the graphical output also reflects the program recursive behavior.

```
program    tri s, t, u
draw    s, t, s+u÷2, t+u, s, t      ←
case
 u > 50
tri    s+u÷7, t+u÷10, u÷3
tri    s+4*u÷7, t+u÷10, u÷3
tri    s+u÷3, t+u÷2, u÷3
esac
end
```

Example 5. Example of recursive program. An arrow traces the statement execution.

The entire presentation of the language up to this point should take less than one-half hour. The student is now able to experiment by programming her/his own recursive programs. A suggestion is made to the student of an interesting pattern that might be programmed; then the student enters the editor.

IV. CONCLUSION

Three points are important to remember. First, within one hour of instruction a student can have an effective tool for experimentation with recursive programming. The language is simple and yet meets all the requirements for generating clear examples. The generation of the output gives some feeling for what recursion is; the instructor has a viable mechanism for presentation.

Secondly, the entire lesson is versatile for use by the student. S(he) can read the textual material, write programs, review more material or watch programmed examples in any order that s(he) wishes. The index suggests to the student the order preferred, but anything can be accessed by the student when s(he) wishes.

Lastly, the language is designed merely to accommodate recursive programming. The language is brief; the functions of controlling recursion through the program use statement and case statement are stressed. Through experimentation, students can discover that certain types of patterns can only be drawn recursively.

A computer-assisted instruction system with a flexible graphics terminal is a viable method for teaching introductory computer science topics. Recursion can be isolated from other topics; and its lesson can be presented at the student's own pace. By eliminating irrelevant material, such as iterative programming techniques, the impact of recursion will have a better lasting effect. Later, the instructor can transfer the

technique learned in pl/2d to other programming languages and still maintain continuity because the examples have a similar frame of reference: they are both programming languages.

LIST OF REFERENCES

[1] Alpert, D. and D. L. Bitzer, "Advances in Computer Based Education," Science, Vol. 167 (20 March 1970), pp. 1582-1590.

[2] LISP 1.5 Programmer's Manual, McCarthy, John, Second Edition, Cambridge, Mass., MIT Press, 1966.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-647	2.	3. Recipient's Accession No.
4. Title and Subtitle pl/2d: A Programming Language for Generating 2-Dimensional Patterns			5. Report Date May 1974 6.
7. Author(s) Karen Sebela Lantz			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois			10. Project/Task/Work Unit No. 11. Contract/Grant No. Sloan AP FDN
12. Sponsoring Organization Name and Address Alfred P. Sloan Foundation 630 Fifth Avenue New York, New York 10020			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

pl/2d is a recursive mini-language which produces a graphical output. The languages allow the student to "see" how recursion works. The motivation for the design was to produce a useful tool that is easy to use and isolates the topic of recursion.

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price